OIPE JC41 MAR 2 5 2004 PATENT & TRADEMARK OFFICE

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Mail Stop Appeal Brief – Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on March 23, 2004.

Signed: [illegible]
Ramin Mahboubian

PATENT #20 Appeal Brief 3-30-04

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Ex Parte Field et al.

Application for Patent
Filed: March 31, 2000
Application No.: 09/540,576
Group Art Unit 2122

Examiner Eric B. Kiss

For: DEBUGGER PROTOCOL GENERATOR

# APPEAL BRIEF

Beyer Weaver & Thomas, LLP
P.O. Box 778
Berkeley, CA 94704
Attorneys for Appellant

03/26/2004 MAHMED1 00000084 09540576
01 FC:1402 330.00 OP

TABLE OF CONTENTS

O I P E JC
MAR 2 5 2004
PATENT & TRADEMARK OFFICE

AF 2122
2170

*IN THE UNITED STATES PATENT AND TRADEMARK OFFICE*

| | |
|---|---|
| In re application of: FIELD et al. | Attorney Docket No.: SUN1P252/P4198 |
| Application No.: 09/540,576 | Examiner: KISS, Eric B. |
| Filed: March 31, 2000 | Group: 2122 |
| Title: DEBUGGER PROTOCOL GENERATOR | Confirmation No.: 2536 |

CERTIFICATE OF MAILING
I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as first-class mail on March 23, 2004 in an envelope addressed to the Commissioner for Patents, Mail Stop Appeal Brief-Patents, P.O. Box 1450 Alexandria, VA 22313-1450.

Signed: [illegible]
Sally Zumba

# APPEAL BRIEF TRANSMITTAL
# (37 CFR 192)

RECEIVED
MAR 2 9 2004
Technology Center 2100

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This brief is in furtherance of the Notice of Appeal filed in this case on December 23, 2003. This brief is transmitted in triplicate.

This application is on behalf of a:
☐ Small Entity ☒ Large Entity

Pursuant to 37 CFR 1.17(f), the fee for filing the Appeal Brief is:
☐ $165.00 (Small Entity) ☒ $330.00 (Large Entity)

☒ Applicant(s) hereby petition for a <u>one-month</u> extension of time to under 37 CFR 1.136. If an additional extension of time is required, please consider this a petition therefor.

☒ Applicant(s) believe that no (additional) Extension of Time is required; however, if it is determined that such an extension is required, Applicant(s) hereby petition that such an extension be granted and authorize the Commissioner to charge the required fees for an Extension of Time under 37 CFR 1.136 to Deposit Account No. 500388.

Total Fee Due:

|  |  |
|---|---|
| Appeal Brief fee | $330 |
| Extension Fee (if any) | $110 |
| **Total Fee Due** | **$440** |

☒ Enclosed is Check No. 21945 in the amount of $440.00

☒ Charge any additional fees or credit any overpayment to Deposit Account No. 500388, (Order No. SUN1P252). Two copies of this transmittal are enclosed.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

[illegible]

R. Mahboubian
Reg. No. 44,890

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300

OIPE JC41
MAR 2 5 2004
PATENT & TRADEMARK OFFICE

## 1. REAL PARTY IN INTEREST

The real party in interest is the assignee, Sun Microsystems, Inc.

## 2. RELATED APPEALS AND INTERFERENCES

It is believed that there are no other appeals or interferences which will directly affect or will be directly affected by or have a bearing on the Board's decision in the pending appeal.

RECEIVED
MAR 2 9 2004
Technology Center 2100

## 3. STATUS OF CLAIMS

Currently, claims 20 and 24-32 are pending and all are being appealed. The claims on appeal are reproduced below in Appendix A. Solely in order to reduce the issues before the Board, claims 1-19 and 21-23 have been canceled by an amendment dated March 23, 2004.

## 4. STATUS OF AMENDMENTS

The amendment canceling claims 1-19 and 21-23 is being submitted with this appeal. All other amendments to the claims have been entered.

## 5. SUMMARY OF INVENTION

The present application pertains to generating various debugging components for a debugging system that operates in a multi-platform environment.

In general, "debugging" refers to the process of locating and fixing or bypassing errors ("bugs) in computer program code or the engineering of a hardware device. Debugging a program or hardware device typically starts with identifying a problem, isolating the source of the problem, and then fixing or bypassing the error (or bug) that caused the problem.

Conventional debugging systems are available for debugging conventional programming environments. The Conventional debugging systems are designed to run on a single platform and must generally debug only applications running on the same or a similar platform. As such, prior art debugging systems are suitable for conventional computing programming environments where the same or similar platforms are used. However, multi-platform computing environments which have been more recently developed (e.g., Java$^{TM}$ programming environment) may be implemented across a wide variety of different hardware platforms and operating systems. As such, these programming environments require a multi-platform debugging system that can debug applications running on various platforms. Conventional debugging systems, however, do not address the needs of a multi-platform computing environment (Specification, paragraph 3).

In a multi-platform debugging system, typically a debugger application is written in a first programming language (e.g., Java$^{TM}$ programming language) while the target side (debuggee) is written in another programming language or native code (e.g., C programming language). Both pieces of code, however, need to be compliant with each other and work together to facilitate debugging of a multi-platform computing environment, or the debugging system will fail (Specification, paragraph 4).

Figure 2a illustrates a Java$^{TM}$ Platform Debugger Architecture (JPDA) that supports local and remote debugging using three separate interfaces. The Java$^{TM}$ Platform Debugger Architecture (JPDA) defines a set of interfaces used for generation of debugger applications, namely, a high level Java$^{TM}$ Debug Interface (JDI), a Java$^{TM}$ Debug Wire Protocol (JDWP), and a Java$^{TM}$ Virtual Machine Debug Interface (JVMDI). The JPDA, among other things, provides a solution to the general connection problems encountered in multi-platform environments (Specification, paragraph 23).

As also illustrated in Figure 2a, the multi-platform debugging system has a front-end debugger component ("front-end"). The front-end, among other things, implements a high-level Java$^{TM}$ Debug Interface (JDI) while a back-end debugging component "back-end" agent, among other things, is responsible for communicating requests from the debugger front-end to the debuggee component (e.g., program) which is being debugged. The back-end can communicate with the front-end, for example, over a communications channel using a Java$^{TM}$ Debug Wire Protocol (JDWP) while the back-end communicates with the debuggee Virtual Machine (VM) using the Java$^{TM}$ Virtual Machine Debug Interface (JVMDI) (Specification, paragraph 24).

As a multi-platform debugging system, a Java$^{TM}$ Platform Debugger Architecture (JPDA) provides a multi-component framework that can be used to facilitate debugging of a program in a multi-platform environment. By way of example, a debugger program written in Java$^{TM}$ programming language can be used via a user interface that runs on first virtual machine (designed for a first platform) to debug another program (debuggee) on a second virtual machine (designed for a second platform) utilizing a debugger agent (back-end) that is written in C programming environment. Accordingly, Java$^{TM}$ Platform Debugger Architecture (JPDA) can be used for debugging applications in a multi-platform system that utilizes various programming environments and languages.

In addition to providing a useful architecture for debugging a multi-platform system, the present application further addresses compatibility and document problems that are encountered in multi-platform systems (Specification, paragraph 30). In accordance with one aspect of the invention, a formal specification language for a multi-platform debugging environment is defined. As will be appreciated, the same formal specification language is used to automatically generate compatible code for both the front-end and back-end components, as well as the documentation needed to implement a multi-platform debugging system in accordance with another aspect of the invention (Specification, paragraph 30).

As illustrated in the embodiment depicted in Fig. 3, a "JDWPGen" program is provided which can be used to automatically generate various components of a multi-platform debugging system. The "JDWPGen" program parses a formal specification which is provided as (JDWP.spec) in accordance with one embodiment of the invention. Based on the (JDWP.spec) received as input, the "JDWPGen" program automatically generates: (1) a protocol documentation (JDWPdetails.html), (2) a front-end JDWP processing module in Java$^{TM}$ programming language (JDWP.java), and (3) a C language "include" file (JDWPConstants.h) which controls the behavior of the back-end JDWP processing module. Since both the

JDWP.java and JDWPConstants.h are generated from the same specification, it is much easier to "debug" the debugger code, and to produce new versions of the JDWP without having to manually re-write two separate programs (Specification, paragraph 33).

In one embodiment, the formal specification language has been developed in a purely declarative language implemented as a Java$^{TM}$ Debug Wire Protocol (JDWP) formal specification language with syntax, grammar, and semantics that are suitable for a multi-platform debugging environment and also facilitate automatic generation of various debugging components in that environment using the same generator (e.g., JDWPGen). The syntax, grammar, and semantics of this formal specification language is illustrated on page 13-15 of the specification.

A JDWPGen employs a recursive descent parser to parse a JDWP formal specification language in accordance with one embodiment of the invention. The parser constructs an abstract syntax tree representation of the formal specification where each node in the abstract syntax tree represents an object encapsulating actions that are needed to generate output for that node. The nodes can correspond directly with statements in the input specification. Further processing can be accomplished by "walking" this abstract syntax tree. Typically, several passes are used to, among other things, resolve names and check for errors. Finally, the tree is walked three more times to generate output: once to generate the Java$^{TM}$ class which is used by the front-end to send and receive information across JDWP, another time to generate the C include file containing the definitions used by the back-end to send and receive information across JDWP, and once more to generate the published human-readable specification document in HTML (Specification, paragraph 36).

In summary, the claimed invention recites a method for automatically generating, based on the same formal specification, front-end code and back-end code debugger components of a multi-platform debugger system in multi-platform computing environment. The code is generated such that these debugging components are compatible with each other. The front-end component provides a high-level debugging Interface associated with platform independent programming language supported on a first virtual machine (front-end). The back-end component provides a virtual machine debugging interface associated with a programming specific code that provides the ability to communicate with a second virtual machine (back-end). Both components use a high-level debugging communication protocol provided to facilitate communication between the first and second virtual machines in the multi-platform computing environment.

## 6. ISSUES

(1) The only issue presented on appeal is whether claims 20 and 24-32 are taught or suggested by material disclosed by the Applicant ("slideshow") in an Information Disclosure Statement dated October 30, 2002 in view of "Official Notice" taken by the Examiner based on Aho et al., "Compilers, Principles, Techniques, and Tools," 1986, Addison Wesley (*"Aho et al."*)

## 7. GROUPING OF CLAIMS

The claims on appeal (20 and 24-32) do not stand or fall together. Each of the claims on appeal will be argued individually.

## 8. ARGUMENT

### A. INTRODUCTION

In the Final Office Action, the Examiner rejected claims (20 and 24-32) under 35 U.S.C. § 102(b) based on a public use or on a sale bar, or in the alternative, under 35 U.S.C. § 103(a) as being obvious over "Java$^{TM}$ One 1998 Presentation," a presentation made on March 26, 1998 in a public forum (hereinafter referred to as "slideshow").

It will be shown that the Examiner's rejection is improper and should be withdrawn. In addition, it will be shown that claims 20 and 24-32 are patentable over the cited art ("slideshow") for several reasons.

First, it will be shown that the Examiner's rejection of 20 and 24-32 is improper because it is not supported by the cited art ("slideshow").

Second, it will be shown that the cited art ("slideshow") does not teach or suggest the features recited in independent claim 20.

Third, it will be argued that the Examiner has not established a *prima facie* case of obviousness because the Examiner has not relied on factual evidence, or general knowledge, aside from conventional compilation techniques, to reject the claimed invention under 35 U.S.C 103(a).

Finally, it will be respectfully submitted that cited reference ("slideshow") and conventional compilation techniques which the Examiner has relied on, taken alone, or in any proper combination do not teach or suggest several other features that are recited in dependent claims 24-32.

## B. THE REJECTION OF CLAIMS 20 AND 24-32 IS IMPROPER BECAUSE THE REJECTION IS NOT SUPPORTED BY THE CITED ART

In the Final Office Action dated September 24, 2003, the Examiner rejected claims 20 and 24-32 under 35 U.S.C. § 102(b) based on a public use or on a sale bar, or in the alternative, under 35 U.S.C. § 103(a) as being obvious over "Java$^{TM}$ One 1998 Presentation," a presentation made on March 26, 1998 in a public forum (hereinafter referred to as "slideshow"). The "slideshow" was submitted by the Applicant in an Information Disclosure Statement dated October 30, 2002, and is also provided in Appendix B for the Board's convenience.

Initially, it is respectfully submitted that the Examiner has made several assertions that are not supported by the "slideshow." For example, the Examiner has asserted that the "slideshow" teaches "a formal specification defining a communication protocol written in Java$^{TM}$ Debug Wire Protocol (JDWP) declarative specification language" (Office Action, page 16, paragraph 12).

To support this assertion, the Examiner has made reference to pages 1, 2, 4 and <u>13</u> of the "slideshow" (Office Action, page 16, paragraph 12). Clearly, pages 1, 2 and 4 of the "slideshow" do not teach or suggest a formal specification defining a communication protocol written in a debug wire protocol. Instead, a very high level overview for Debugging <u>support</u> tools is illustrated. This overview announces as future plans a "Full Jbug

Implementation" ("Slideshow", page 1, "Future JDK 1.2"). A "partial Jbug implementation" (Initial JDK 1.2) is illustrated on page 2 of the "slideshow." A very high-level black-box diagram illustrates the Future JDK 1.2 Debugging support ("Slideshow", page 4, "Future JDK 1.2"). It should be noted that the slideshow consists of only nine (9) pages (B1-B9 reproduced in Appendix B). The Examiner's assertion <u>cannot</u> possibly be supported by page thirteen (13) of the "slideshow" because the "slideshow" does not have a thirteen (13). Hence, contrary to the Examiner's assertion, pages 1, 2 and 4 of the "slideshow" do not teach a formal specification defining a communication protocol written in Java$^{TM}$ Debug Wire Protocol (JDWP) declarative specification language." In other words, the Examiner's rejection is not supported by the cited reference. Accordingly, it is respectfully submitted that the rejection of claims 20 and 24-32 is improper, and it is respectfully requested that the Board consider directing the Examiner to withdraw this rejection.

Furthermore, it is respectfully submitted that the Examiner has made several other assertions that are clearly NOT supported by the cited reference ("slideshow"). For the Board's convenience, some of the Examiner's assertions that are not supported by the "slideshow" are reproduced below:

- sending events generated in the second virtual machine to the front-end via the back-end debugger program code portion (see, for example, pages 3-6 and 13-19 of *Slideshow*);
- the front-end reading and parsing events from the back-end debugger code portion (see, for example, pages 3-6 and 13-19 of *Slideshow*);
- the front-end processing module performing operations related to requests made through the front-end debugger program by the debugger application program (see, for example, pages 3-6 and 13-19 of *Slideshow*;
- the front-end processing module writing formatted requests (see, for example, pages 3-6 and 13-19 of *Slideshow*);
- the back-end processing module performing operations related to event processing and request processing (see, for example, pages 3-6 and 13-19 of *Slideshow*);

- the event processing operations including sending an event which was generated through the virtual machine debugging interface to the front-end debugging portion (see, for example, pages 3-6 and 13-19 of *Slideshow*);

- the request processing operations including reading and parsing formatted requests from the front-end debugger program portion (see, for example, pages 3-6 and 13-19 of *Slideshow*);

- the front-end debugger program portion including a class which is used by the front-end debugger program portion to send and receive information over the debugging communication protocol (see, for example, pages 3-6 and 13-19 of *Slideshow*).

Clearly, the "slideshow" does not teach or suggest any of these features and the Examiner's rejection is improper for these additional reasons. Again, it is very respectfully requested that the Board direct the Examiner to withdraw this improper rejection.

## C. THE CITED ART ("SLIDE SHOW") DOES NOT TEACH OR SUGGEST THE FEATURES RECITED IN THE INDEPENDENT CLAIM 20

It is noted that the "slideshow" illustrates a very high-level debugging architecture that includes a front-end and a back-end which are depicted as black boxes ("slideshow," page 7). Furthermore, it is noted that the "slideshow" states that a Java$^{TM}$ Debug Wire Protocol ("JDWP") can be defined as a protocol for communication between the front-end and back-end ("slideshow," page 8). This protocol would allow the front and back ends to run on different Virtual Machines (VM) which are respectively implemented of different platforms. To provide a very high-level overview for those not skilled in the art, the "slideshow" also notes that the JDWP (like other communication protocols) can describe how information is exchanged at the low-level as bits (1's and 0's) over a medium (wire). In other words, JDWP can be referred to as a "specification of bits on the wire" ("slideshow", page 8).

However, it is respectfully submitted that "slideshow" does not teach or suggest a <u>formal specification</u> for a debug communication protocol in a multi-platform environment. This formal specification is provided in accordance with one aspect of the invention. For example, in one embodiment, the formal specification is a purely declarative language that was developed for the Java$^{TM}$ Debug Wire Protocol (JDWP formal specification). This formal specification is illustrated in pages 13-15 of the specification.

The "slideshow," however, merely illustrates that a communication protocol (JDWP) can facilitate communication between a front-end and back-end in a very high-level black-box architectural model. In other words, from a very high-level black-box architectural view it is illustrated that if a debugging system is divided into a front-end and a back-end then a high-level communication protocol can be used to facilitate communication. The "slide show", however, does no teach or suggest how this high-level communication protocol can be implemented. More importantly, the "slideshow" does not teach or suggest a <u>formal specification</u> for a debugging communication protocol. Furthermore, it is respectfully submitted that a formal specification for a debugging communication protocol that connects front-end and back-end bugging components were not disclosed in the "Java$^{TM}$ One conference". Accordingly, claims 20 and 24-32 are patentable for at least this reason alone.

Moreover, it is respectfully submitted that the "slideshow" does not teach or even remotely suggest <u>automatically</u> generating a front-end debugger program portion <u>from a formal specification</u> in the context of the invention. Clearly, the "slideshow" does not teach or suggest this feature as evidenced by a through examination of all nine (9) pages of the "slideshow." In addition, the Applicant respectfully submits that <u>automatically</u> generating a front-end debugger program portion <u>from the formal specification</u> was not disclosed in the "Java$^{TM}$ One conference". Thus, it is respectfully submitted that claims 20, 24-32 are patentable over the "slideshow" for this additional reason.

Furthermore, it is respectfully submitted that the "slideshow" does not teach or suggest <u>automatically</u> generating a <u>back-end</u> debugger program portion <u>from a formal specification</u> in the context of the invention. In addition, the Applicant respectfully submits that <u>automatically</u> generating a back-end debugger program from a formal specification was not disclosed in the "Java$^{TM}$ One conference." Thus, it is respectfully submitted that claims 20, 24-32 are patentable for yet an additional reason.

Still further, the "slideshow" does not teach or suggest automatically generating both the front-end and back-end components of a debugging system based on the same formal specification. It is further submitted that the combination of these features were not disclosed in "Java$^{TM}$ One conference." Thus, it is respectfully submitted that claims 20, 24-32 are patentable for still another reason.

Finally, it is respectfully submitted that it should be evident that the "slideshow" does not teach or suggest: inputting a formal specification into a code generator, and parsing the formal specification in order to automatically generate the front-end and back-end debugging components that are compatible with each other and comply with the formal specification (claim 20).

Accordingly, it is respectfully submitted that claims 20, 24-32 are patentable over the "slideshow," and it is very respectfully requested that the Board direct the Examiner to allow these claims in order to avoid further delay in issuance of the case.

## D. THE EXAMINER HAS NOT ESTABLISHED A *PRIMA FACIE* CASE OF OBVIOUSNESS BECAUSE THE EXAMINER HAS NOT RELIED ON FACTUAL EVIDENCE, OR GENERAL KNOWLEDGE, ASIDE FROM CONVENTIONAL COMPILATION TECHNIQUES, TO REJECT THE CLAIMED INVENTION

In the Final Office Action, the Examiner has noted that it is <u>unclear</u>, based on materials made available to the Examiner, whether the "slideshow" teaches (a) <u>inputting</u> a formal specification into a code generator, (b) <u>parsing</u> the formal specification, and (c) <u>generating</u> a Java$^{TM}$ front-end debugger program portion and back-end Java$^{TM}$ debugger program portion from the formal specification after parsing the formal specification (Office Action, page 18).

Initially, it is respectfully submitted that the Examiner has failed to meet his burden of establishing a *prima facie* case of obviousness because, among other things, the Examiner has NOT relied on factual evidence to make a rejection under 35 U.S.C § 103(a). The Examiner has merely indicated that it is <u>unclear to him</u> whether these features are taught by the "slideshow" and has done so despite the Applicant's statements on the record that these same features were not disclosed in the "Java$^{TM}$ One conference" (please see, for example, amendment dated March 20, 2003, page 3, listing these features among other features that were not disclosed in "Java$^{TM}$ One conference").

Nevertheless, despite the Applicant's statements on the record and any factual evidence to the contrary, the Examiner has summarily rejected these recited features by taking "Official Notice" that in order to arrive at a machine-readable implementation of a human readable specification, a compilation process comprising parsing a specification and generating output are well known and commonly practiced in the computer art as evidenced by an introductory book on compilers which is often used in undergraduate studies in computer science (Final Office Action, pages 18-19, citing Alfred V. *Aho et al.*, "Compilers, Principles, Techniques and Tools," 1986).

Clearly, the "Official Notice" taken by the Examiner does NOT even address a particular claimed feature (e.g., automatically generating various debugger components based on the same formal specification) in a meaningful way since the Applicant has not broadly claimed using a compilation process to compile a human readable specification (e.g., Fortran programming language) to arrive at a machine-readable implementation (e.g., binary code). Again, the Applicant has stated on the record that these features (a, b and c above) were not disclosed in the "Java™ One conference," and the Examiner has not pointed to any factual evidence that may even suggest otherwise. Accordingly, it is respectfully submitted that the Examiner has failed to establish a *prima facie* case of obviousness for this reason alone.

Furthermore, even assuming entirely for the sake of argument that a communication protocol between a front-end and back-end debugging component is known, it does not necessarily follow that it would have been obvious to one skilled in the art at the time the invention was made to use conventional compiling techniques of *Aho et al.* in order to <u>automatically</u> generate <u>both</u> front-end and back-end <u>debugger</u> components based on the <u>same formal</u> specification that is also compliant with the communication protocol used by these components to communicate with each other. Per MPEP § 2143, there must be some suggestion or motivation for this automation. The Examiner, however, has not pointed to a suggestion or motivation for combining or modifying the "slideshow" and *Aho et al.* In fact, the Examiner has not even pointed to some suggestion or motivation for automatically generating a single debugging component based on a formal specification.

To establish a *prime facie* case of obviousness, The Examiner must show a motivation or suggestion for <u>automatically</u> generating <u>both</u> a front-end and a back-end-debugger component. Also, there is a need for a motivation or suggestion to formalize a specification or develop a formal specification to achieve this automation.

Clearly, the Examiner has not met this burden because the Examiner has NOT even pointed to a motivation or suggestion for automatically generating a single debugging component based on a formal specification. Instead, the Examiner has merely asserted that the alternative to compiling is writing code directly in assembly language or binary which is typically impractical (Final Office Action, page 19).

As a side note, it is respectfully submitted that there are other "practical" alternatives aside from writing the code directly in assembly language or binary to generate the front-end and/or back-end debugging components in the context of the claimed invention. For example, each of the front-end and back-end components may be manually generated using a different specification. In other words, it is not required as the only alternative to writing code directly in assembly language or binary that the same formal specification be used to automatically generate both the front-end and back-end components. In fact, prior to the invention, the front-end and back-end of the multi-platform debugger were not both automatically generated based on the same formal specification. As noted in the specification, the code for the back-end processing module was manually written (Specification, paragraph 33). But this doesn't mean that this code had to be written in assembly or binary code. As noted in the specification, a "include" file for the back-end was manually written in a high-level programming language (e.g., the C programming language) (Specification, paragraph 33).

Thus, contrary to the Examiner's assertion, not having to code in binary or assembly language is NOT a motivation or suggestion for automatically generating both back-end and front-end debugging components from the same formal specification. One motivation or suggestion for this automation is noted in the specification, namely, a desire to address compatibility and documentation issue in an effort to promote the evolution of multi-platform debugging systems (Specification, paragraph 33). To achieve this, a formal specification was conceived which, among other things, can be used to automatically generate both the front-end and back-end debugging components in the context of the claimed invention.

Accordingly, it is very respectfully submitted that the Examiner may have misconstrued what may be a practical alternative to the automation in the context of the claimed invention and, as a result, the Examiner has failed to provide a motivation or suggestion for the automation features of the claimed invention. In any case, clearly, the Examiner has NOT met his burden to establish *prima facie* case of obviousness for at least these additional reasons.

Still further, it is respectfully submitted that in order to establish a *prima facie* case of obviousness, all claim limitations must be taught or suggested by the cited art (MPEP § 21403.03). However, the combination of the "slideshow" and conventional compiling techniques do not teach or suggest these claimed limitations. As noted above, the "slideshow" does not teach many of the recited features of the claimed invention. As will be discussed, the conventional compiling techniques of *Aho et al.* also fail to teach or suggest these features. In fact, these conventional compiling techniques do not even address the specific problems encountered in multi-platform debugging environments.

Also, other conventional techniques fail to address these needs. As noted in the specification, languages exist for the specification of inter-process/object communication, such as the Interface Definition Language (IDL) which is part of the Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG). These languages are compiled (i.e., by an IDL compiler) to produce stubs for the client side of communication and skeletons for the server side. However, such languages also fail to address the problems associated with generating protocol compliant debugger code (Specification, paragraph 5).

The conventional compiling techniques of *Aho et al.* fail to even address specific problems that are encountered in multi-platform debugging environments and as such <u>cannot</u> possibly teach or suggest many of the claimed features in the context of the invention. First, it should also be noted that multi-platform debugging environments introduce several problems that cannot merely be addressed by general conventional compilation techniques which the Examiner has relied on. Typically, a front-end component (debugger) is written in one programming language (e.g., Java$^{TM}$ programming language) while the back-end (debuggee) is written in another (e.g., C programming language). Yet both of these components are to be compliant with the same formal specification and are to be automatically generated based on the same formal specification in accordance with the claimed invention (see, for example, claim 20).

In addition, it is respectfully submitted that the conventional compiling techniques of *Aho et al.* do NOT even address these claimed features because, among other things, a conventional compiler of *Aho et al.* produces code only for a single programming language that is, in turn, executed on a particular platform (hardware and/or operating system). In other words, conventional compiling techniques do not teach or suggest generating different programming codes for different components based on the same input.

Also, conventional compiling techniques of *Aho et al.* relied on by the Examiner do not teach or suggest generating debugging components that communicate with each other in accordance with a specific communication protocol. In fact, conventional compiling techniques of *Aho et al.* generally teach generating binary or assembly code for a high-level programming language. These techniques, however, do not teach producing program code for a platform independent component (e.g., front-end debugger code written in Java$^{TM}$ programming language), or a definition file for a platform dependent programming language ( e.g., "Constant.h" file for a back-end written in C programming language), or a human readable specification in a markup language (e.g., HTML documents). Moreover, the conventional compiling techniques of *Aho et al.* do not teach or suggest automatically generating all of these components from a single specification.

Clearly, conventional compiling techniques of *Aho et al.* relied on by the Examiner merely teach general parsing and compiling of a program written in a programming language to generate binary or assembly code. Conventional compiling techniques, however, do not teach or suggest: automatically generating, based on the same specification, various debugger components that communicate with each other. Hence, even assuming purely for the sake of argument that a communication protocol for a multi-platform debugging environment was known, one of ordinary skilled in the art at the time of invention could NOT merely adapt the introductory compiling principles of *Aho et al.* to automatically generate a front-end debugger that provides a high-level debugging interface in a platform independent language and a back-end debugger program associated with a specific platform from the same formal specification, such that these components are compatible with each other and comply with the same formal specification, and yet function and meet the requirements of a debugging environment using a high level debugging communication protocol provided for communication between first and second virtual machines that respectively support the front-end and back-end debugger components (claim 20).

To achieve the claimed features, a purely declarative language with syntax, grammar and semantics suitable for multi-platform debugging systems can be used in accordance with one embodiment of the invention. The suitable syntax, grammar and semantics of a Java™ Debugger Wire protocol is illustrated on pages 13-15 of the specification in accordance with one embodiment of the invention. Clearly, introductory compiling techniques of *Aho et al.* do NOT teach or suggest suitable syntax, grammar and semantics for a debugging wire protocol or any other reasonable technique for automatically generating a front-end debugger program portion and a back-end debugger program that are generated from the same formal specification in the context of the claimed invention.

Accordingly, it is respectfully submitted that the rejection of claims under U.S.C. § 103 (a) is improper, and it is respectfully requested that the Board consider directing the Examiner to withdraw all rejections under U.S.C. § 103(a).

## E. THE "SLIDESHOW" AND *AHO ET AI.,* TAKEN ALONE OR IN ANY PROPER COMBINATION, DO NOT TEACH OR SUGGEST MANY OTHER FEATURES THAT ARE RECITED IN THE DEPENDENT CLAIMS 24-32

In view of the foregoing arguments, it is earnestly believed that is evident that the "slideshow" and *Aho et al.,* taken alone or in any proper combination, do NOT teach or suggest many other claimed features recited in dependent claims 24-32. Therefore, without further discussion, it is respectfully submitted that the "slideshow" and *Aho et al.,* taken alone or in any proper combination, do not teach or suggest many other recited features of dependent claims 24-32.

These features, for example, include the following: a front-end processing module that operates to send events that are generated in a virtual machine to a front-end debugger program portion via a back-end debugger program code portion (claim 24); a front-end processing module that performs one or more of the following operations: read and parse events from the back-end debugger code portion, convert the events from a first format into a second format which is compatible with a front-end debugger code portion, and queue the events (claim 25); a front-end processing module that performs operations related to requests made through a front-end debugger program by a debugger application program (claim 26); a front-end processing module that performs one or more of the following operations: write formatted requests, send the formatted requests to the back-end debugger code portion, associate at least one reply with the formatted requests, and read and parse the at least one reply, and deliver the at least one reply to an appropriate requester (claim 27); and a front-end debugger program portion that includes a class which is used by the front-end debugger program portion to send and receive information over the debugging communication protocol (claim 32).

These features also include: a back-end processing module that performs operations related to event processing and request processing (claim 28), or performs event processing operations including: sending an event which was generated through a virtual machine debugging interface to a front-end debugging portion (claim 29), or such that the request processing operations include one or more of the following operations: reading and parsing formatted requests from a front-end debugger program portion, forwarding the requests to the back-end debugger program code portion, and sending the reply to the requests to the front-end debugger program portion (claim 30).

Accordingly, It is respectfully submitted that the rejection of claims 24-32 is improper, and it is respectfully requested that the Board consider directing the Examiner to allow these claims in order to avoid further delay in issuance of the case.

## 9. CONCLUSION

In view of the foregoing, it is respectfully submitted that the Examiner's rejection of claims 20 and 24-32 over the cited art is erroneous. Accordingly, it is respectfully requested that the Board consider reversing these claims and direct the Examiner to allow the claims in order to avoid further delay in issuance of these claims.

Respectfully submitted,
BEYER WEAVER & THOMAS, LLP

[illegible]

R. Mahboubian
Registration No. 44,890

P.O. Box 778
Berkeley, CA 94704-0778
(650) 961-8300

## 10. APPENDIX A

### CLAIMS ON APPEAL

20. (Previously Presented) A method of providing a debugging environment in a computing environment that includes first and second virtual machines, the method comprising:

inputting a formal specification written in a debugging specification language into a program code generator, the formal specification defining a high level debugging communication protocol for communication between the first and second virtual machines;

parsing the formal specification using the program code generator;

automatically generating a front-end debugger program portion from the formal specification based on the parsing of the formal specification, the front-end debugger program running on a first virtual machine, the front-end debugger program portion corresponding to a platform independent programming language which provides a high level debugging interface which can be accessed by a debugger application operating on a first virtual machine;

automatically generating a back-end debugger program code portion from the formal specification based on the parsing of the formal specification, the back-end debugger program code portion implementing a virtual machine debugging interface which provides the capability to control and communicate with a second virtual machine, the back-end debugger program code portion corresponding to a platform-specific programming language; and

wherein the front-end debugger program portion and the back-end debugger program code that are generated from the formal specification are compatible with each other and comply with the formal specification, thereby implementing the high-level communication protocol between the first and second virtual machines.

21. (Canceled)

22. (Canceled)

23. (Canceled)

24. (Previously Presented) A method as recited in claim 20, wherein the front-end processing module operates to send events that are generated in the second virtual machine to the front-end debugger program portion via the back-end debugger program code portion.

25. (Previously Presented) A method as recited in claim 24, wherein the front-end processing module performs one or more of the following operations:

read and parse events from the back-end debugger code portion;

convert the events from a first format into a second format which is compatible with the front end debugger code portion; and

queue the events.

26. (Previously Presented) A method as recited in claim 24, wherein the front-end processing module further performs operations related to requests made through the front-end debugger program by the debugger application program.

27. (Previously Presented) A method as recited in claim 24, wherein the front-end processing module further performs one or more of the following operations:

write formatted requests

send the formatted requests to the back-end debugger code portion;

associate at least one reply with the formatted requests;

read and parse the at least one reply;

deliver the at least one reply to an appropriate requester.

28. (Previously Presented) A method as recited in claim 27, wherein the back-end processing module performs operations related to event processing and request processing.

29. (Previously Presented) A method as recited in claim 28, wherein the event processing operations performed by the back-end processing module includes sending an event which was generated through the virtual machine debugging interface to the front-end debugging portion.

30. (Previously Presented) A method as recited in claim 29, wherein the request processing operations performed by the back-end processing module include one or more of the following operations:

reading and parsing formatted requests from the front-end debugger program portion;

forwarding the requests to the back-end debugger program code portion;

sending the reply to the requests to the front-end debugger program portion.

31. (Previously Presented) A method as recited in claim 20, wherein the back-end processing module performs operations related to event processing and request processing.

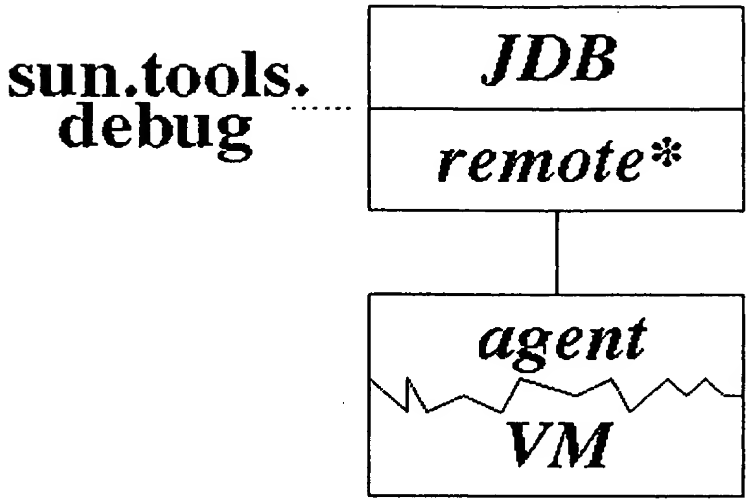
32. (Previously Presented) A method as recited in claim 20, wherein the front-end debugger program portion includes a class which is used by the front-end debugger program portion to send and receive information over the debugging communication protocol.

# APPENDIX B

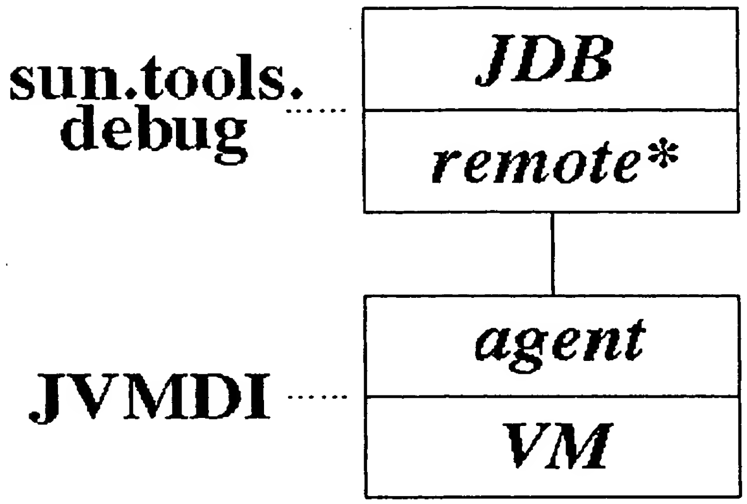
## Debug Support: Overview

- JDK 1.1
  - JDB & sun.tools.debug
- Initial JDK 1.2
  - Partial Jbug Implementation (JVMDI)
  - Retrofitted JDB & sun.tools.debug
- Future JDK 1.2
  - Full Jbug Implementation
  - Graphical Debugger Application

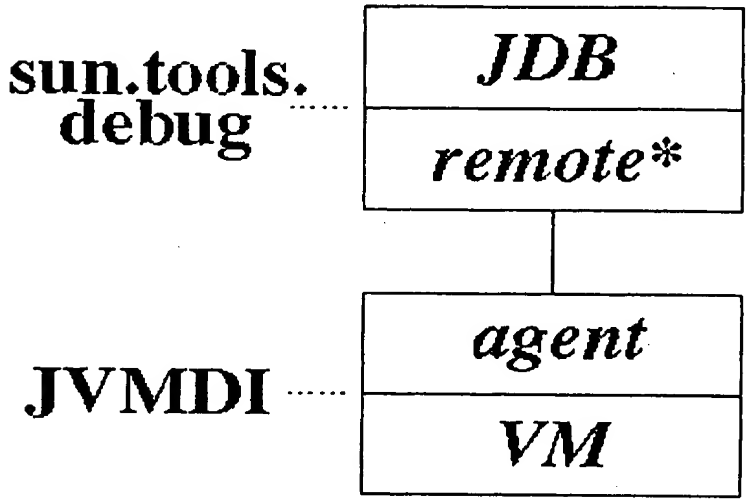
# sun.tools.debug (JDK 1.1)



# sun.tools.debug (JDK 1.2)



# sun.tools.debug (JDK 1.2)



## Java Virtual Machine* Debug Interface (JVMDI)

- Implemented by VM
- Native "C" interface
- Low Level
- Works with and extends JNI

***As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.**

# Initial JDK 1.2

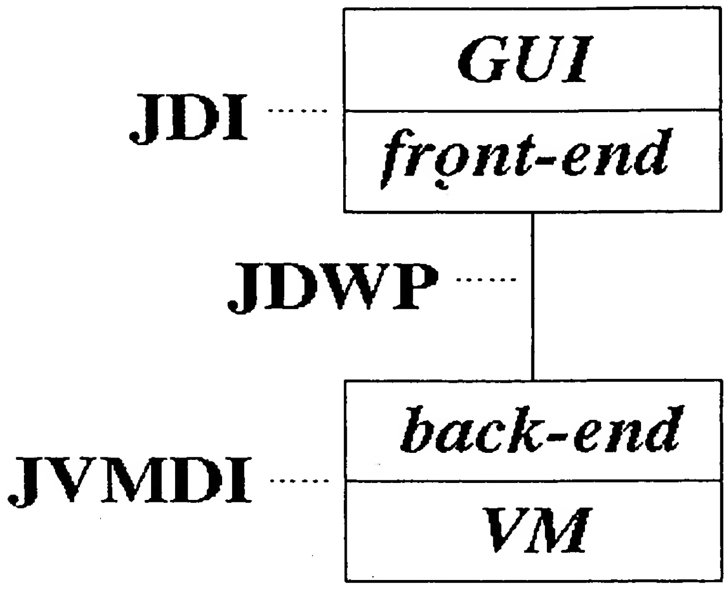
Improvements

- Alternative interface (JVMDI)
- A way to plug in JVMs (JVMDI)
- Bug fixes

Unaddressed Problems

- Frequent hangs
- Class loading anomalies
- Unfriendly line-mode application
- Many basic debugger functions missing

# JBUG: Architecture



## Java Debug Wire Protocol (JDWP)

- Specification of bits on the wire
- Transport (socket, serial, ...) independent
- Defines communication between front and back ends
- Allows front/back end to run different VM/Platform

## Java Debug Interface (JDI)

- Java Language interface
- High Level
- Allows 100% Pure Java debugger implementations
- More robust, complete than sun.tools.debug